Hyper-Threading Technology and its Impact on OpenMP*

Agenda

➡ • Hyper-Threading Overview

• Exploiting Hyper-Threading Technology

  – Explicit Threads

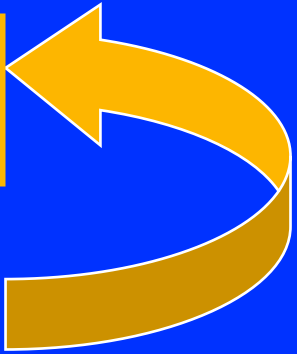
  – OpenMP Programming API

• OpenMP* Programming Example

Today’s Processors

- Single Processor Systems
  - Instruction Level Parallelism (ILP)
  - Performance improved with more CPU resources

Hyper-Threading technology enables TLP to single processor system.

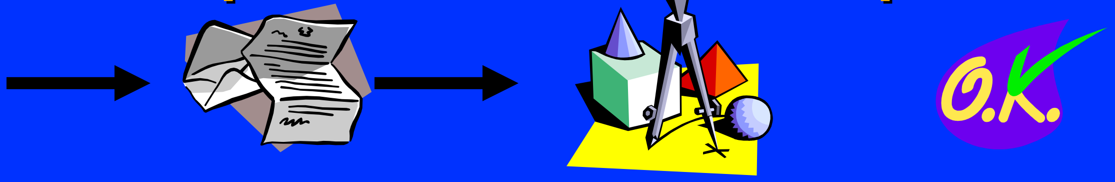


- Multiprocessor Systems
  - Thread Level Parallelism (TLP)
  - Performance improved by adding more CPUs

Hyper-Threading Overview

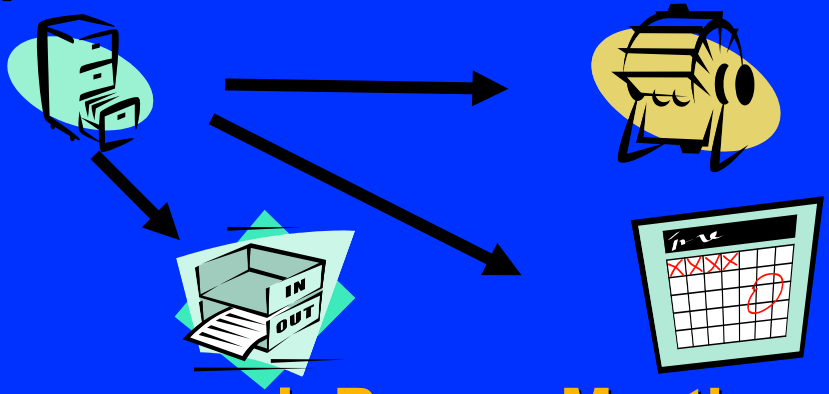
Today's Software

- Sequential tasks



Open File → Edit → Spell Check

- Parallel tasks



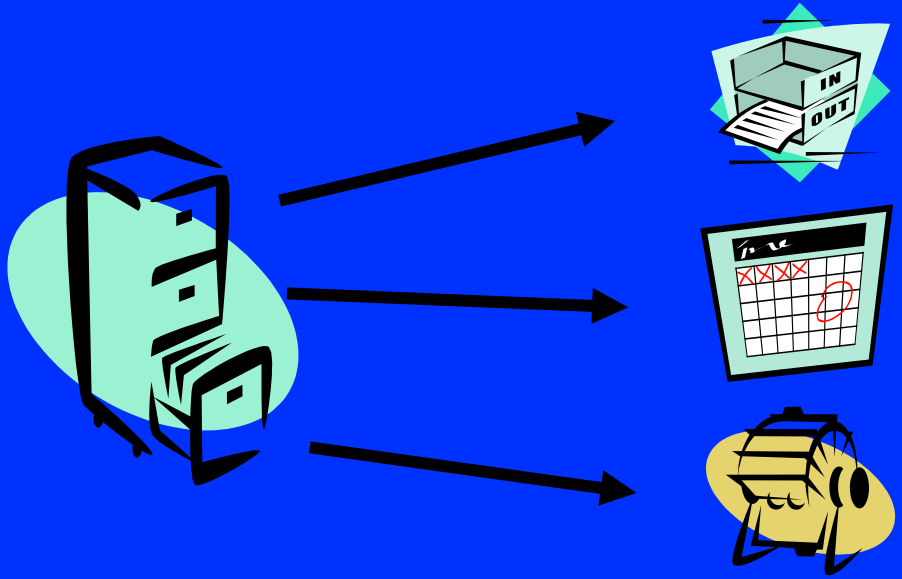
Open DB's → Address Book

Open DB's → InBox

Open DB's → Meeting

Multi-Processing

- Run parallel tasks using multiple processors



CPU 1

CPU 2

CPU 3

Multi-tasking workload + processor resources
=> Improves MT Performance

Hyper-Threading Overview

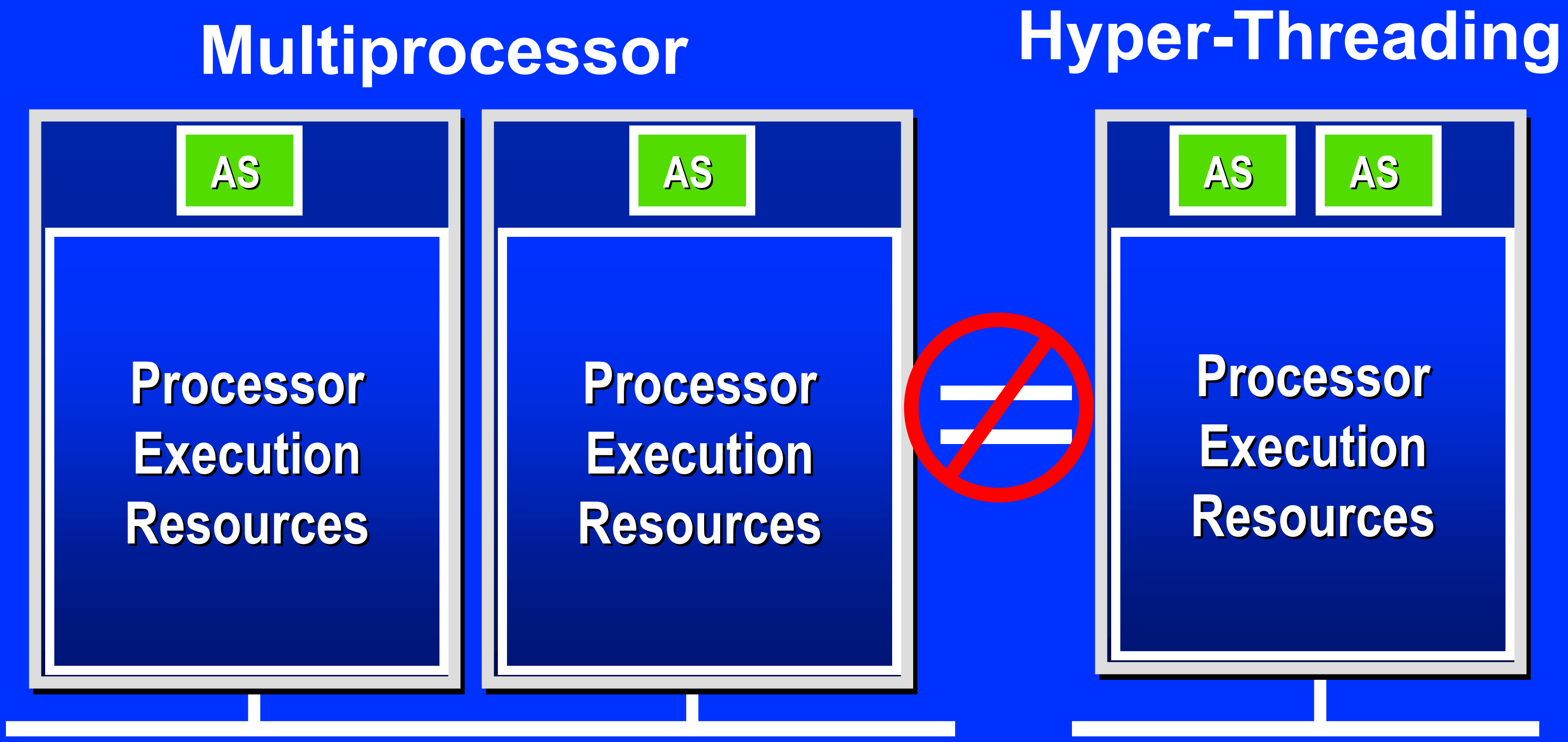
The Increase in Instruction Processing (Multi-tasking Workload from Previous Slide) Throughput of Hyper-Threading is Due to:

- The design of the Intel Netburst Micro-architecture
- The mix of IA-32 Instructions typically found in multi-threaded code

Why Hyper-Threading?

- Studies[1] show that only 35% of execution resources of the Intel NetBurst Micro-architecture are used
- Hyper-Threading technology takes advantage of the inherent parallelism of multithreaded code to provide the processor core with a second thread of execution

Hyper-Threading Technology


Multiprocessor
Hyper-Threading
AS
Processor Execution Resources
AS
Processor Execution Resources
AS
AS
Processor Execution Resources

AS = Architecture State (eax, ebx, control registers, etc.), xAPIC

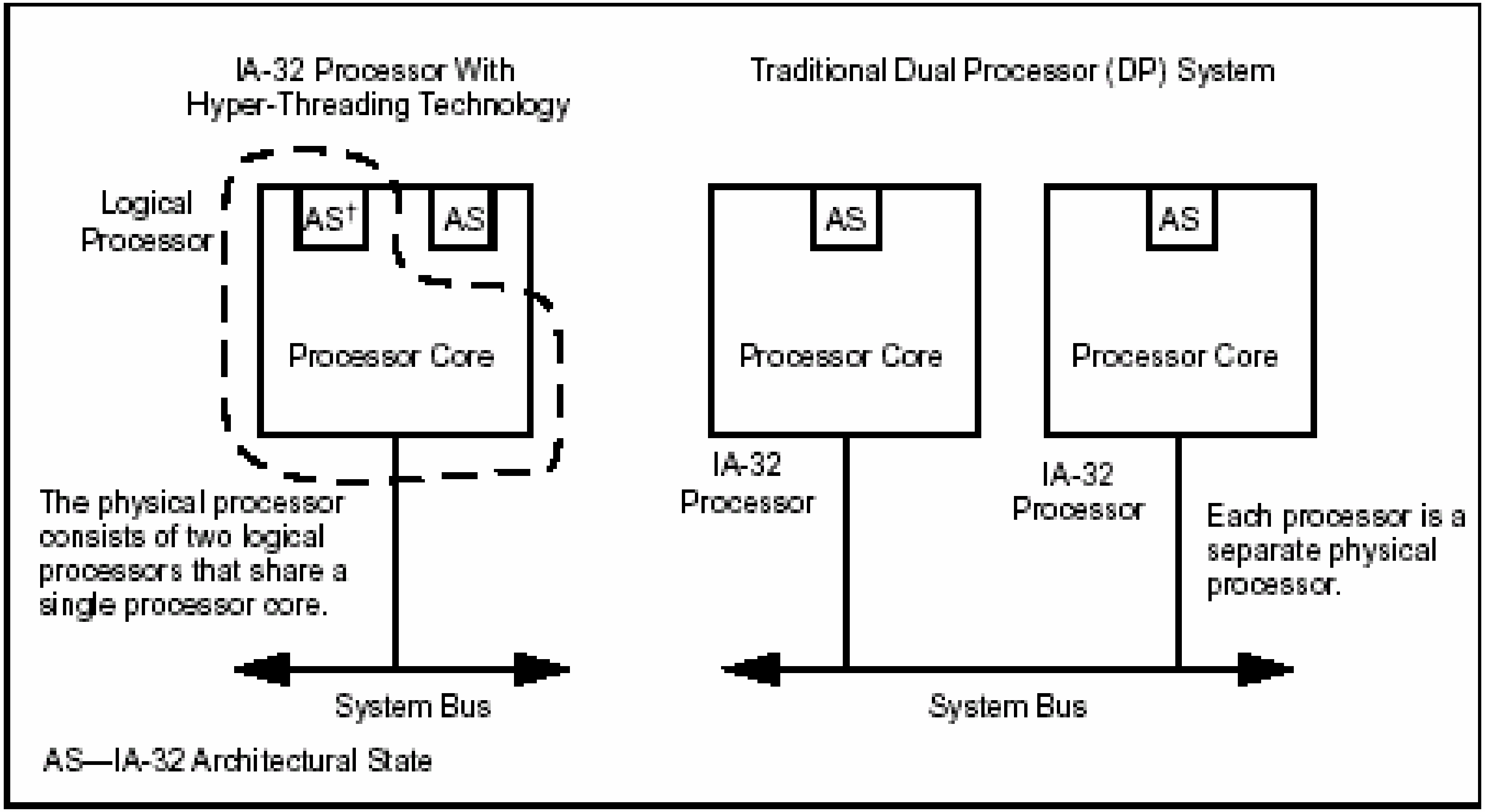
Hyper-Threading Technology looks like two processors to software

How is this done?

- Processors that are enabled with Hyper-Threading Technology:
  - Manage incoming instructions from two different software threads
  - The processor keeps track of the data processing status of each set of instructions

Hyper-Threading Overview

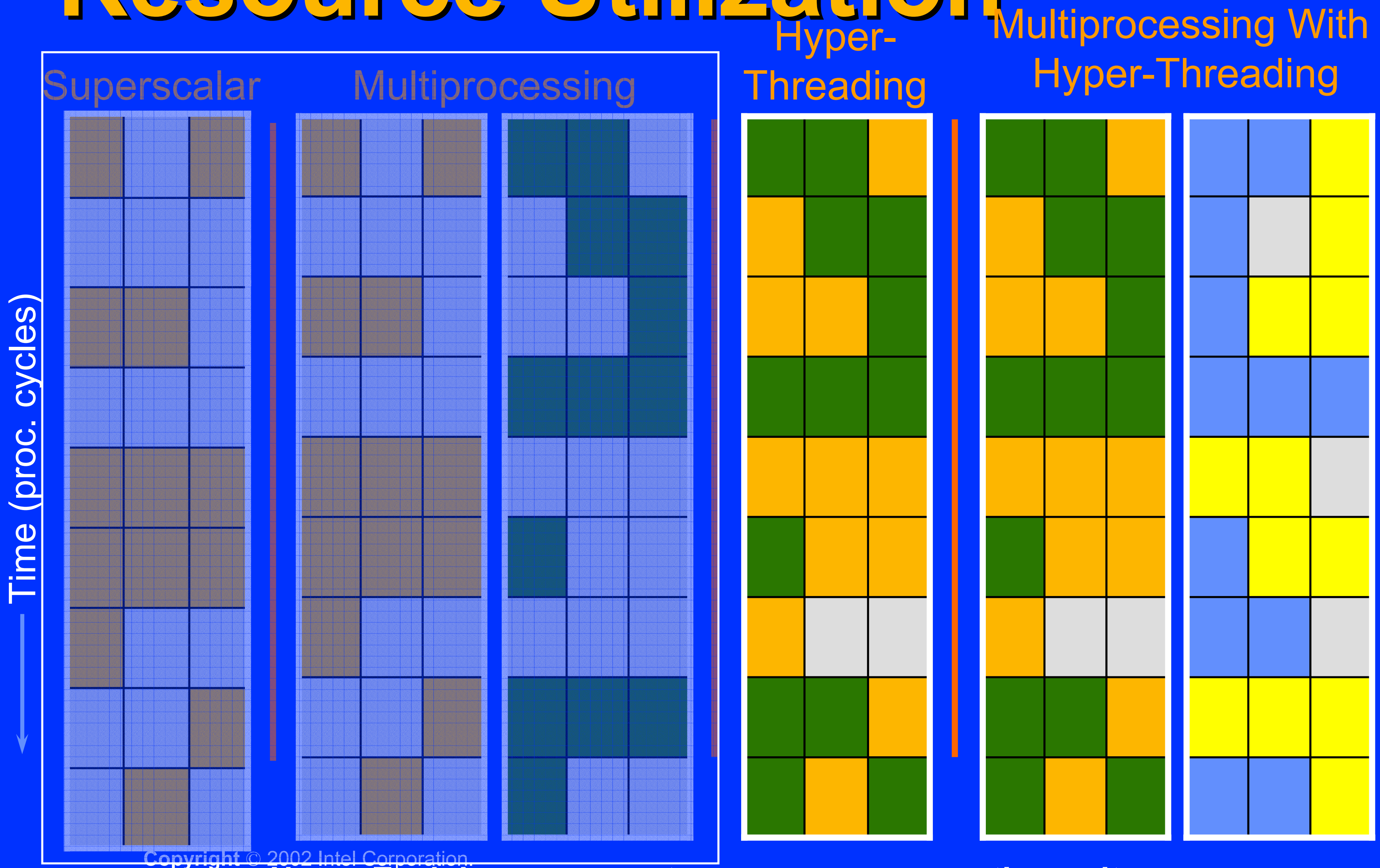
Switching from One Logical Processor to the Other [1]


IA-32 Processor With Hyper-Threading Technology
Traditional Dual Processor (DP) System
Logical Processor
AS†
AS
Processor Core
AS
Processor Core
AS
Processor Core
IA-32 Processor
IA-32 Processor
The physical processor consists of two logical processors that share a single processor core.
Each processor is a separate physical processor.
System Bus
System Bus
AS—IA-32 Architectural State

Three Intel Xeon^{TM} Design Goals to Support Hyper-Threading[2]

- Minimize Die Area – less than 5% in additional die area cost
- When one logical processor stalls the other logical processor continues to make forward progress
- A single threaded application running on a processor with Hyper-Threading technology executes at same speed as a processor without this capability

Resource Utilization



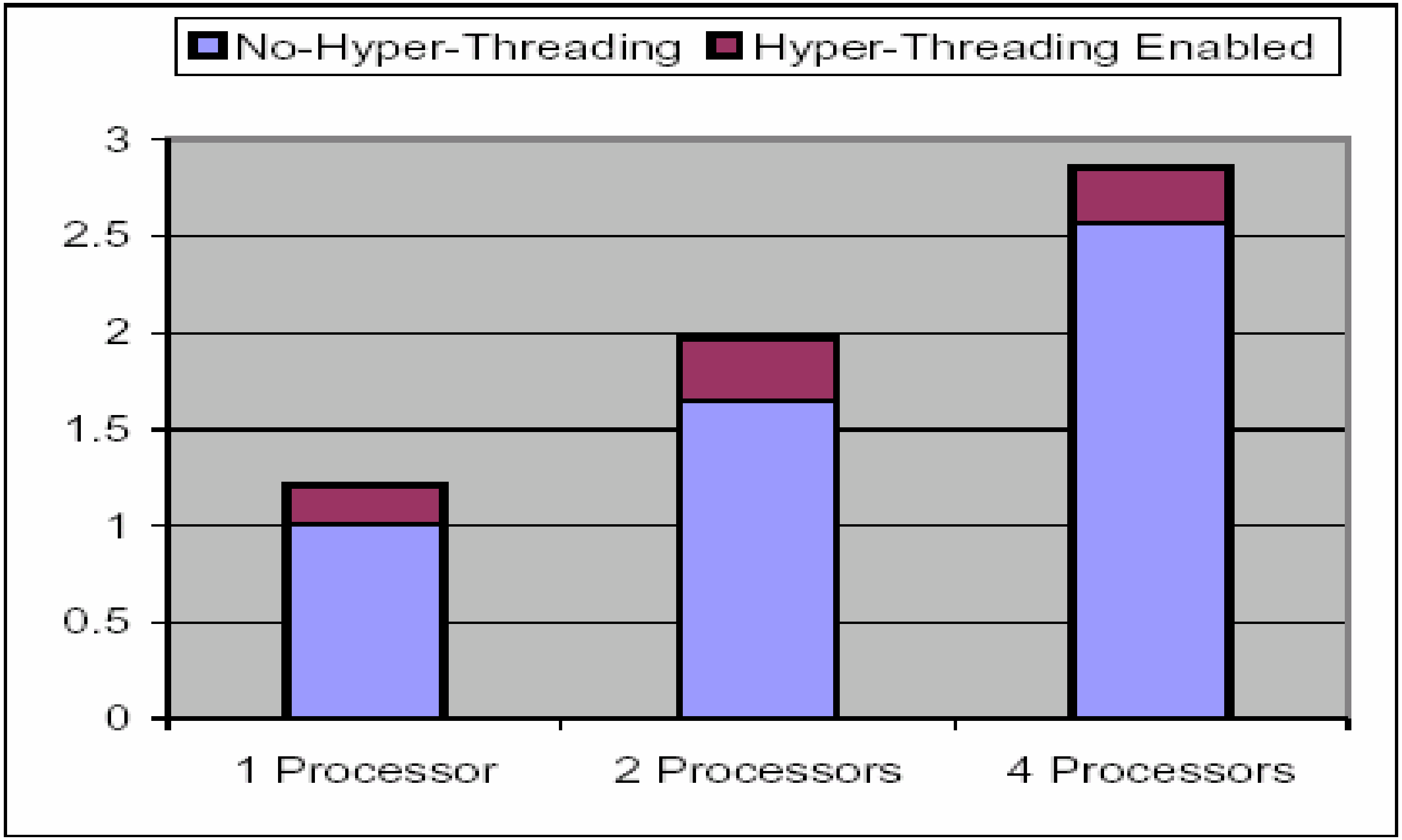
Note: Each box represents a processor execution unit

Key Point

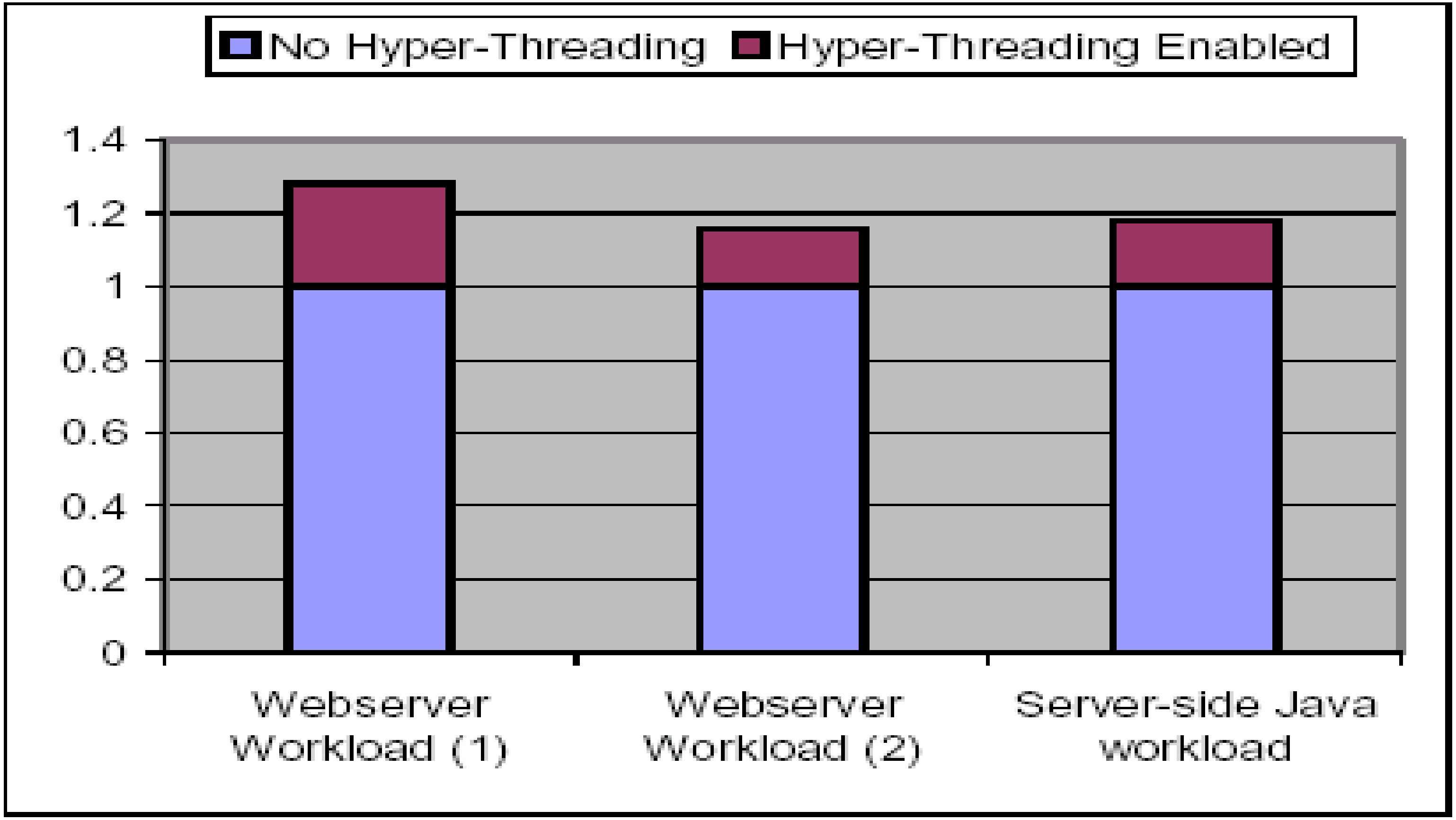
- Hyper-Threading technology enables better utilization of hardware resources
- Hyper-Threading technology provides more computing power for multi-threaded applications

Hyper-Threading Overview

Performance Increases from Hyper-Threading Technology on an Online Transaction Processing Workload[2]



Web Server Benchmark Performance[2]

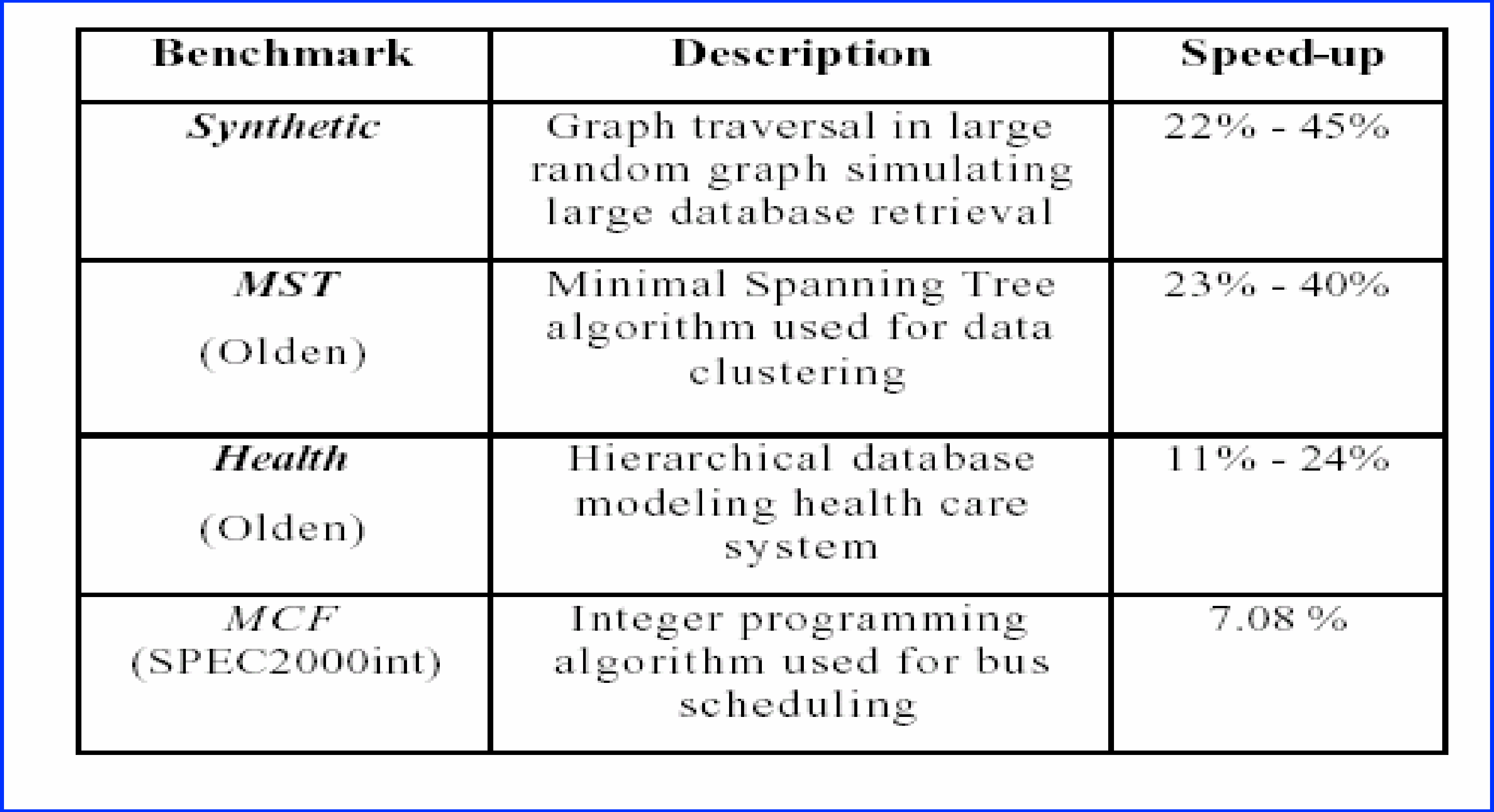


Software-based Speculative Precomputation[3]

- Technique to improve the latency of single-threaded applications
- Algorithmic Sketch:
  - Speculative thread fetches memory objects (usually in a strided manner)
  - Main thread does the computation with prefetched data objects
- Acronym - SP

Hyper-Threading Overview

Initial performance data: speculative prefetching (SP) on a pre-production version of an Intel® Xeon™ processor with Hyper-Threading Technology[3]



Benchmark	Description	Speed-up
Synthetic	Graph traversal in large random graph simulating large database retrieval	22% - 45%
MST (Olden)	Minimal Spanning Tree algorithm used for data clustering	23% - 40%
Health (Olden)	Hierarchical database modeling health care system	11% - 24%
MCF (SPEC2000int)	Integer programming algorithm used for bus scheduling	7.08 %

Performance Gains with Hyper-Threading Technology

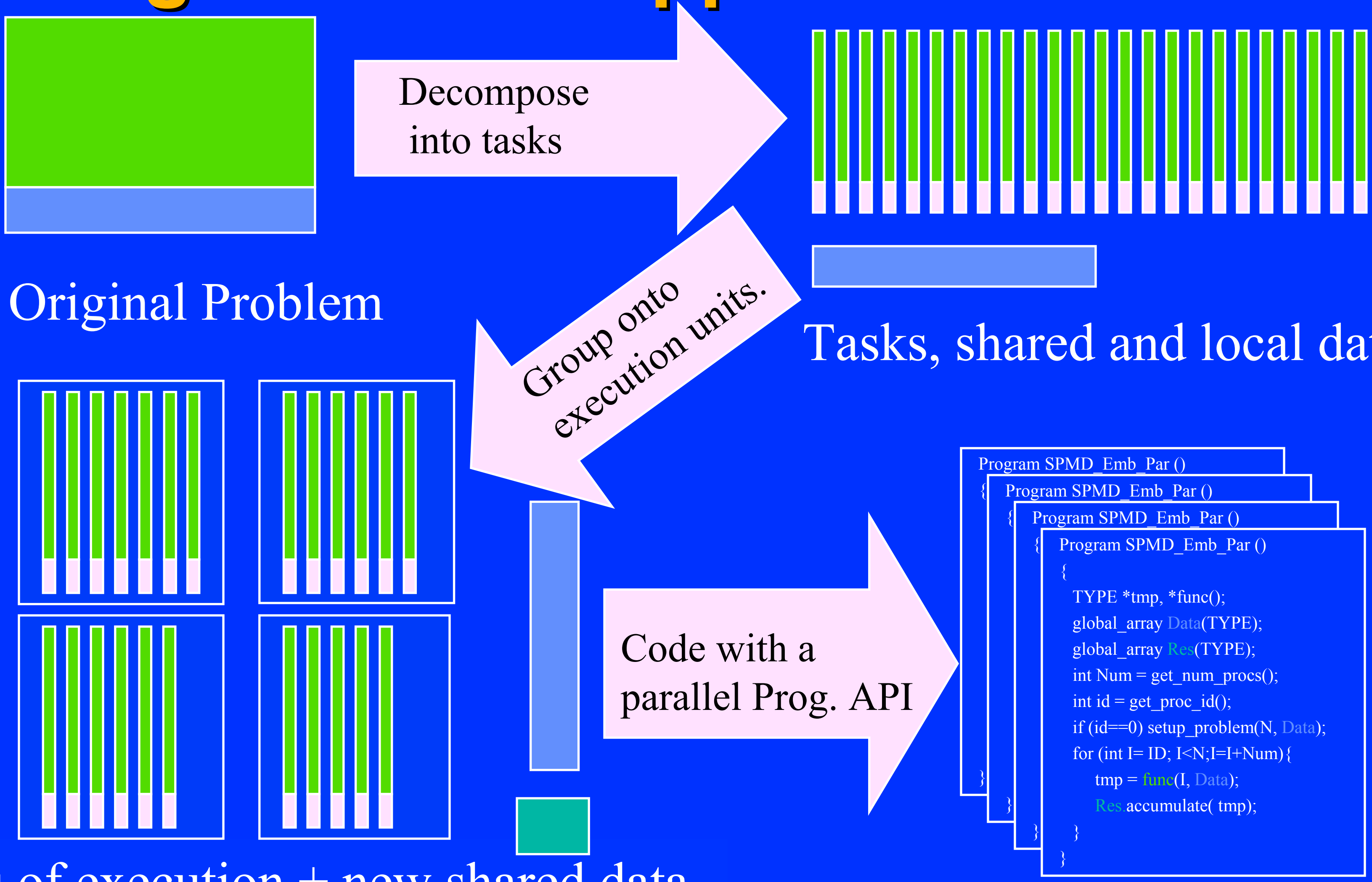
- Hyper-Threading technology can provide a performance gain of up to 30% over a comparable IA-32 processor without Hyper-Threading technology, assuming:
  - Multithreaded operating system and application code
- For multiprocessor systems:
  - Increase in computing power will generally scale linearly with an increase in the number of physical processors
- Scalability of performance is highly dependent on the nature of the application

Agenda

- Hyper-Threading Overview
- ➡ Exploiting Hyper-Threading Technology
  - Explicit Threads
  - OpenMP Programming API
- OpenMP* Programming Example

Parallel Computing Overview

Writing a Parallel Application



Decompose into tasks

Original Problem

Tasks, shared and local data

Group onto execution units.

Code with a parallel Prog. API

```
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
{
  TYPE *tmp, *func();
  global_array Data(TYPE);
  global_array Res(TYPE);
  int Num = get_num_procs();
  int id = get_proc_id();
  if (id==0) setup_problem(N, Data);
  for (int I= ID; I<N;I=I+Num){
      tmp = func(I, Data);
      Res.accumulate( tmp);
  }
}
```

Units of execution + new shared data for extracted dependencies

be cla Corresponding source code

Parallel Computing Overview

What is OpenMP*?

```
C$OMP FLUSH
#pragma omp critical
C$OMP THREADPRIVATE(/ABC/)
CALL OMP_SET_NUM_THREADS(10)
C$
)
C$
C
RED
#
C$OMP PARALLEL COPYIN(/blk/)
C$OMP DO lastprivate(XX)
Nthrds = OMP_GET_NUM_PROCS()
omp_set_lock(lck)
```

OpenMP: An API for Writing Multithreaded Applications*

– **Compiler directives and library routines for parallel application programmers**

– **Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++**

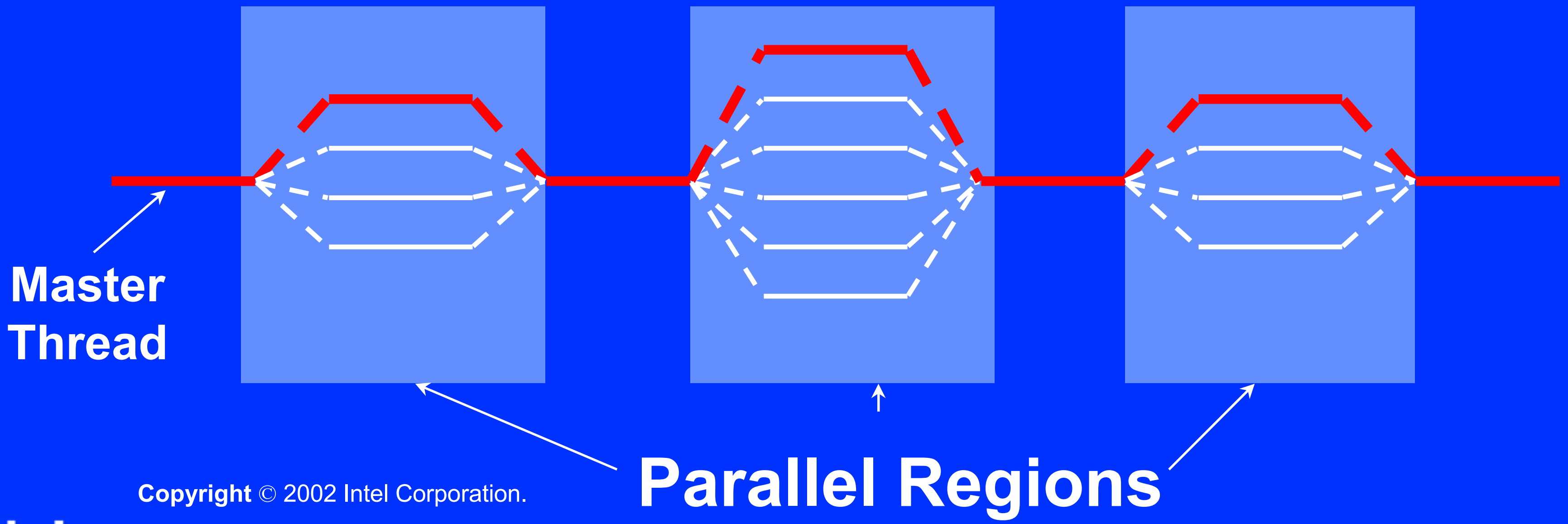
– **Standardizes last 15 years of SMP practice**

*Other names and brands may be claimed as the property of others.

OpenMP* Programming Model

Fork-Join Parallelism:

- Master thread spawns a team of threads as needed.
- Parallelism is added incrementally: i.e., the sequential program evolves into a parallel program.



*Other names and brands may be claimed as the property of others.

Agenda

- Hyper-Threading Overview
- Exploiting Hyper-Threading Technology
  - Explicit Threads
  - OpenMP Programming API
- → OpenMP* Programming Example

*Other names and brands may be claimed as the property of others.

Pi Program

```
static long num_steps = 100000;
double step;
void main ()
{          int i;   double x, pi, sum = 0.0;

           step = 1.0/(double) num_steps;

           for (i=1;i<= num_steps; i++){
                   x = (i-0.5)*step;
                   sum = sum + 4.0/(1.0+x*x);
           }
           pi = step * sum;
}
```

Pi: Windows* Threads

```
#include <windows.h>
#define NUM_THREADS 2
HANDLE thread_handles[NUM_THREADS];
CRITICAL_SECTION hUpdateMutex;
double global_sum = 0.0;

void Pi (void *arg)
{
   int i, start;
  double x, sum = 0.0;
  static long num_steps = 100000;
  double step;


  start = *(int *) arg;
  step = 1.0/(double) num_steps;

  for (i=start;i<= num_steps; i=i+NUM_THREADS){
       x = (i-0.5)*step;
      sum = sum + 4.0/(1.0+x*x);
  }
  EnterCriticalSection(&hUpdateMutex);
  global_sum += sum;
  LeaveCriticalSection(&hUpdateMutex);
}
```

```
void main ()
{
  double pi; int i;
  DWORD threadID;
  int threadArg[NUM_THREADS];

  for(i=0; i<NUM_THREADS; i++)   threadArg[i] = i+1;

  InitializeCriticalSection(&hUpdateMutex);

  for (i=0; i<NUM_THREADS; i++){
            thread_handles[i] = CreateThread(0, 0,
                                (LPTHREAD_START_ROUTINE) Pi,
                                &threadArg[i], 0, &threadID);
}

  WaitForMultipleObjects(NUM_THREADS,
                                thread_handles, TRUE,INFINITE);

  pi = global_sum * step;

  printf(" pi is %f \n",pi);
}
```

Doubles code size!

Simple Is Better

Threads libraries:

- Pro: Programmer has control over everything
- Con: Programmer must control everything

Control over all threads → High complexity → High programming costs

The simplicity of OpenMP* lowers programming costs.

*Other names and brands may be claimed as the property of others.

Pi: OpenMP* version

```
#include <omp.h>
static long num_steps = 100000;         double step;

void main ()
{          int i;     double x, pi, sum = 0.0;
           step = 1.0/(double) num_steps;

#pragma omp parallel for reduction(+:sum) private(x)
           for (i=1;i<= num_steps; i++){
                     x = (i-0.5)*step;
                     sum = sum + 4.0/(1.0+x*x);
           }
           pi = step * sum;
}
```

OpenMP* adds 2 lines of code.

OpenMP*: Easy as Pi

```
#include <omp.h>
static long num_steps = 100000;         double step;
void main ()
{           int i;     double x, pi, sum = 0.0;
            step = 1.0/(double) num_steps;
#pragma omp parallel for reduction(+:sum) private(x)
            for (i=1;i<= num_steps; i++){
                      x = (i-0.5)*step;
                      sum = sum + 4.0/(1.0+x*x);
            }
            pi = step * sum;
}
```

OpenMP* simplifies multithreading.

Key Take Away

If a multi-threaded application performance does not improve on an MP system, you will have minimal benefit with Hyper-Threading technology.

Summary

- Hyper-Threading Technology gives you more computing power to throw at your problems.
- OpenMP* is an easy to use API for writing multithreaded programs.
- Continue to use good threaded programming practices with Hyper-Threading technology.

References

- [1] “Introduction to Hyper-Threading Technology” at the URL: http://www.intel.com/technology/hyperthread/download/25000802.pdf .
- [2] D. T. Marr, F Binns, D. L. Hill, G. Hinton, D. Koufaty, J. A. Miller, M. Upton, “Hyper-Threading Technology Architecture and Microarchitecture”, *Intel Technology Journal,* Vol. 6, No. 1, February 2002.
- [3] H. Wang, P. Wang, R. D. Weldon, S. M. Ettinger, H. Saito, M. Girkar, S. S. Liao, J. P. Shen, “Speculative Precomputation: Exploring the Use of Multithreading for Latency”, *Intel Technology Journal,* Vol. 6, No. 1, February 2002.

Call to Action

- Think of Hyper-Threading as a Technology to Improve Instruction Throughput of Processors
- Can Benefit Multiprocessor Applications, e.g. OpenMP*

*Other names and brands may be claimed as the property of others.